Actividad: Manejo de Estado Global y Persistente en React

Esta actividad se entregará el lunes 19 de mayo a las 23:59pm. Deben trabajar en esto el día de hoy lunes, el jueves, y/o en sus casas. Las clases serán para dudas de esta actividad, para dudas del proyecto, pueden dividir la clase en trabajar en el proyecto y en esta actividad.

Contexto y objetivos

En esta sección del proyecto aprendimos a manejar estado global y persistente sin autenticación real, usando herramientas modernas de React y Next.js. Nuestro objetivo fue:

- Permitir que un usuario invitado marque películas como favoritas.
- Persistir esa información entre visitas y recargas.
- Mostrar una lista personalizada de favoritas en una página dedicada.
- Usar buenas prácticas como separación de servicios, uso de contextos, y UI reactiva.

Este flujo es similar al de muchas aplicaciones reales donde no hay login obligatorio, pero se ofrece una experiencia personalizada.

Implementación con Next.js y TMDb API

En este proyecto trabajamos con **Next.js (App Router)** para construir una aplicación de películas que se conecta a la API de *The Movie Database (TMDb)*. Uno de los objetivos es permitir que los usuarios marcaran películas como favoritas y poder recuperar esa información más adelante, incluso sin una cuenta real. Para esto, implementamos una solución con sesión de invitado, manejo de favoritos, almacenamiento local y estado global.

Para gestionar y compartir el estado de la sesión de invitado en toda la aplicación, utilizamos **React Context**. Un *context* nos permite evitar el "prop drilling" (pasar props manualmente a muchos niveles), ofreciendo un acceso directo al estado desde cualquier componente dentro del árbol. En este caso, creamos un *GuestSessionContext* que guarda el guestSessionId, lo que nos permite acceder a él desde cualquier página o componente que lo necesite.

Servicio de autenticación de invitado

```
// src/services/auth/getGuestSession.ts

import api from "../api";

export const getGuestSession = async () => {
  try {
    const { data } = await api.get("/authentication/guest_session/new");
    return data;
  } catch (error) {
    throw error;
  }
};
```

```
// src/providers/GuestSessionContext.tsx

"use client";

import { createContext, useContext, useEffect, useState } from "react";
import { getGuestSession } from "@/services/auth/getGuestSession";

const GuestSessionContext = createContext<{
  guestSessionId: string | null;
  setGuestSessionId: (id: string) => void;
}>({
  guestSessionId: null,
  setGuestSessionId: () => {},
});

export const GuestSessionProvider = ({
  children,
}: {
  children: React.ReactNode;
}) => {
  const [guestSessionId, setGuestSessionIdState] = useState(
    null
  );
```

```
// src/providers/GuestSessionContext.tsx

//continuación

  const setGuestSessionId = (id: string) => {
    localStorage.setItem("guestSessionId", id);
    setGuestSessionIdState(id);
  };

  useEffect(() => {
    const existingId = localStorage.getItem("guestSessionId");
    if (existingId) {
      setGuestSessionIdState(existingId);
    } else {
      fetchGuestSession();
    }
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, []);

  const fetchGuestSession = async () => {
    const data = await getGuestSession();
    if (data.guest_session_id) {
      setGuestSessionId(data.guest_session_id);
    }
  };

  return (

      {children}

  );
};

export const useGuestSession = () => useContext(GuestSessionContext);
```

Implementación del layout principal

```
// src/app/layout.tsx

import { GuestSessionProvider } from "@/providers/GuestSessionContext";

// Código existente
return (
   <html lang="en">
     <body className={`antialiased`}>
       <GuestSessionProvider>
         <Header />
         <main className="p-6 mt-16">{children}</main>
       </GuestSessionProvider>
     </body>
   </html>
 );

// Código existente
```

Además del contexto, utilizaremos **localStorage**, que nos permite guardar información clave en el navegador de forma persistente, incluso si el usuario recarga la página. Lo utilizamos para almacenar el ID de sesión de invitado y para mantener un estado local de las películas marcadas como favoritas, de manera que podamos actualizar la UI rápidamente sin tener que esperar una respuesta de la API. Otras opciones posibles para almacenamiento persistente incluyen cookies, IndexedDB o sincronización con un backend.

Servicios para gestión de favoritos

```
//src/services/accounts/markAsFavorite
.ts

import api from "../api";

export const markAsFavorite = async (
  movieId: number,
  favorite: boolean,
  guestSessionId: string
) => {
  try {
    const body = {
      media_type: "movie",
      media_id: movieId,
      favorite,
    };

    const { data } = await api.post(

`/account/${guestSessionId}/favorite`,
      body
    );

    return data;
  } catch (error) {
    throw error;
  }
};
```

```
//src/services/accounts/getFavoriteMov
ies.ts

import api from "../api";

export const getFavoriteMovies = async
(guestSessionId: string) => {
  try {
    const { data } = await api.get(

`/account/${guestSessionId}/favorite/m
ovies?language=en-
US&page=1&sort_by=created_at.asc`
    );

    return data;
  } catch (error) {
    throw error;
  }
};
```

En cuanto a los servicios, creamos tres funciones clave para interactuar con la API de TMDb: una para obtener el guestSessionId, otra para marcar o desmarcar una película como favorita, y una más para obtener todas las películas marcadas como favoritas por el usuario. Estas funciones utilizan axios para hacer llamadas GET y POST a los endpoints de TMDb, y están separadas en su propio módulo de servicios para mantener un buen orden.

Página de detalle de película

```
//src/app/movie/[id]/page.tsx

// Ver archivo en canvas
```

La página de detalle de película se actualizó para incluir un botón de "Add to Favorites". Este botón consulta el estado actual (si ya es favorito) desde `localStorage`, actualiza la UI inmediatamente cuando el usuario hace clic, y llama a la API para sincronizar el estado remoto. Si se hace clic nuevamente, se remueve de favoritos y se actualiza tanto la UI como el localStorage.

Página de favoritos

```
// src/app/my-favorites/page.tsx

// Ver archivo en canvas
```

Finalmente, creamos una página de "Mis Favoritos", que consulta la API usando el guestSessionId y muestra todas las películas marcadas como favoritas. Si no hay ninguna, la UI muestra un mensaje amigable al usuario. Esta sección aprovecha la reutilización del componente MovieList para mostrar las películas en un grid visual coherente con el resto del diseño.

Tareas adicionales para completar

1

Agregar recomendaciones

Al detalle de película llamar al endpoint **https://api.themoviedb.org/3/movie/{movie_id}/recommendations** y con el resultado de películas mostrarlas en un carrusel.

2

Implementar paginación

Agregar paginado a las páginas de popular, top rated, now playing y favoritos. Se le pasa al endpoint como query param, ejemplo de favoritos: /account/${guestSessionId}/favorite/movies?language=en-US&page=1

3

Completar la página de inicio

Completar home, es libre, se pueden basar en **https://www.themoviedb.org/**. Mínimo llamar a 3 endpoints.

API de referencia: https://developer.themoviedb.org/reference/intro/getting-started

Resultados y aprendizajes

Implementamos un sistema completo de favoritos para usuarios anónimos

Usar React Context para compartir estado global

Aprovechamos localStorage para persistir el estado entre sesiones

Consumimos la API de TMDb con métodos personalizados y ordenados

Construimos una UI interactiva que responde en tiempo real al usuario

Con esta base, ya podríamos escalar a un sistema con autenticación real o extenderlo a otras funcionalidades como listas personalizadas, puntuaciones o comentarios, usando el login que nos proporciona TMDB.